	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorio de Computacion Salas A y B

Profesor(a): Rene Adrian Davila Perez

Asignatura: Programacion orientada a objetos

Grupo: 01

No de practica(s): proyecto 1

Integrante(s): 322001549, 322006829, 322125533, 425131152, 322103351

No de lista o brigada: 07

Semestre: 2026-1

Fecha de entrega: Viernes 26 de septiembre de 2025

Observaciones:

Calificacion:

Índice

1. Introducción

1.1. Planteamiento del Problema

Se debe diseñar un programa que simule de manera simple el funcionamiento de un carrito, ejecutándose directamente en la terminal. Para ello, se requiere crear clases que contengan los atributos y métodos necesarios para gestionar los artículos de forma organizada y eficiente.

1.2. Motivación

Esta práctica se basa en reforzar los principios de la programación orientada a objetos, aplicando conceptos como definición de atributos, uso de estructuras de datos, constructores personalizados y métodos que operen sobre dichas estructuras. Estos conocimientos son esenciales para adquirir experiencia en el diseño de programas modulares y escalables, incluso en aplicaciones sencillas.

1.3. Objetivos

- Crear clase Carrito y la clase principal Proyecto donde estará el método main, con el fin de establecer comunicación entre clases mediante objetos.
- Definir al menos los atributos articulo y precio.
- Definir al menos una estructura de datos para guardar los objetos en la clase Carrito.
- Definir al menos un constructor (diferente al constructor por defecto), para la clase Carrito.
- Poder realizar operaciones para agregar y quitar artículos a la estructura de datos en la clase Carrito.
- Poder imprimir la lista de contenido del carrito de compras.

2. Marco Teórico

2.1. Clase

Una clase en Java define un nuevo tipo de datos que puede incluir campos (variables) y métodos para definir el comportamiento de los objetos creados a partir de la clase: [1]

```
1
2  class ClassName {
3      // Fields
4      dataType fieldName;
5
6      // Constructor
7      ClassName(parameters) {
8          // Initialize fields
9      }
10
11     // Methods
12     returnType methodName(parameters) {
13         // Method body
14     }
15 }
```

De donde donde:

- ClassName: El nombre de la clase.

- fieldName: Variables que contienen el estado de la clase.
- Constructor: Método especial utilizado para inicializar objetos.
- methodName: Funciones definidas dentro de la clase para realizar acciones.

2.2. Objeto

Un objeto es una instancia de una clase. Se crea utilizando la palabra clave new seguida del constructor de la clase: [1]

```

ClassName objectName = new ClassName(arguments);
```

De donde:

- objectName: El nombre del objeto.
- arguments: Valores pasados al constructor para inicializar el objeto

A continuación un ejemplo incluyendo ambos conceptos:

```
class Car {
    // Fields
    String color;
    int year;

    // Constructor
    Car(String color, int year) {
        this.color = color;
        this.year = year;
    }

    // Method
    void displayInfo() {
        System.out.println("Car color: " + color + ", Year: " +
    year);
    }
}

public class Main {
    public static void main(String[] args) {
        // Creating an object
        Car myCar = new Car("Red", 2020);
        myCar.displayInfo(); // Output: Car color: Red, Year: 2020
    }
}
```

En este ejemplo, la clase Car tiene los campos color y year, un constructor para inicializar estos campos, y un método displayInfo() para mostrar los detalles del coche. Se crea un objeto myCar utilizando la clase Car.

2.3. Constructor

En Java, un constructor es un método especial utilizado para inicializar objetos. A diferencia de los métodos normales, los constructores se invocan cuando se crea una instancia de una clase. Tienen

el mismo nombre que la clase y no tienen tipo de retorno. Los constructores son esenciales para establecer los valores iniciales de los atributos del objeto y prepararlo para su uso.[2]

Tipos de constructores:

1. Constructor por defecto: El compilador Java proporciona automáticamente un constructor por defecto si no se definen constructores explícitamente en la clase. Inicializa el objeto con valores por defecto.
2. Constructor sin argumentos: Un constructor sin argumentos es definido explícitamente por el programador y no recibe ningún parámetro. Es similar al constructor por defecto, pero puede incluir código de inicialización personalizado.[2]
3. Constructor parametrizado: Un constructor parametrizado acepta argumentos para inicializar un objeto con valores específicos. Esto permite una inicialización más flexible y controlada de los atributos de los objetos.[2]
4. Copiar Constructor: Un constructor de copia se utiliza para crear un objeto nuevo como copia de un objeto existente. Java no proporciona un constructor de copia por defecto; sin embargo, puede implementarse manualmente.[2]

A continuación sus sintaxis de constructor por defecto:

```
class ClassName {
    // Constructor
    ClassName() {
        // Initialization code
    }

    // Parameterized Constructor
    ClassName(dataType parameter1, dataType parameter2) {
        // Initialization code using parameters
    }

    // Copy Constructor
    ClassName(ClassName obj) {
        // Initialization code to copy attributes
    }
}
```

Ejemplo de constructor:

```
public class Car {
    String model;
    int year;

    // Default Constructor
    public Car() {
        model = "Unknown";
        year = 0;
    }

    public static void main(String[] args) {
        Car car = new Car();
        System.out.println("Model: " + car.model + ", Year: " +
    car.year);
    }
}
```

En este ejemplo, la clase Car tiene un constructor por defecto que inicializa los atributos model y

year con valores por defecto. Cuando se crea un objeto Car, se le asignan estos valores por defecto. [2]

Ejemplo de constructor parametrizado:

```
public class Car {
    String model;
    int year;

    // Parameterized Constructor
    public Car(String model, int year) {
        this.model = model;
        this.year = year;
    }

    public static void main(String[] args) {
        Car car = new Car("Toyota", 2021);
        System.out.println("Model: " + car.model + ", Year: " +
    car.year);
    }
}
```

Ejemplo de sobrecarga de constructores:

```
public class Car {
    String model;
    int year;

    // No-Argument Constructor
    public Car() {
        model = "Unknown";
        year = 0;
    }

    // Parameterized Constructor
    public Car(String model, int year) {
        this.model = model;
        this.year = year;
    }

    public static void main(String[] args) {
        Car car1 = new Car();
        Car car2 = new Car("Honda", 2022);
        System.out.println("Car1 -> Model: " + car1.model + ",
    Year: " + car1.year);
        System.out.println("Car2 -> Model: " + car2.model + ",
    Year: " + car2.year);
    }
}
```

Este ejemplo demuestra la sobrecarga de constructores en la clase Car, donde se definen tanto un constructor sin argumentos como un constructor parametrizado. Esto permite crear objetos con valores predeterminados o específicos.[2]

2.4. Métodos, parámetros y argumentos

2.4.1. Métodos

Los métodos son bloques de código que se utilizan para realizar tareas específicas. Cada método tiene un nombre que lo identifica y puede recibir una serie de parámetros, los cuales son variables

que proporcionan información necesaria para que el método realice su tarea. Estos parámetros actúan como entrada para el método, permitiéndole operar con los valores recibidos.[3]

Dentro del cuerpo del método, se pueden realizar diversas operaciones, como cálculos matemáticos, manipulación de datos o incluso llamadas a otros métodos. Una vez que el método ha completado sus operaciones, puede devolver un resultado utilizando la palabra clave return seguida del valor que se desea devolver.[3]

La capacidad de reutilizar código es una de las ventajas clave de los métodos en Java. Al definir un método una vez, podemos llamarlo en diferentes partes de nuestro programa, evitando la repetición innecesaria de código. Esto no solo simplifica nuestro código, sino que también facilita su mantenimiento y permite una mayor modularidad en nuestra aplicación.[3]

```
public tipo_de_dato_devuelto nombre_metodo(tipo_de_dato parametro1
    , tipo_de_dato parametro2) {
    // codigo a ejecutar
    return resultado;
}
```

2.4.2. Parámetros

Son las variables que se definen en la declaración de un método y sirven para recibir información externa. Actúan como entradas que el método necesita para ejecutar sus instrucciones.[3]

```
public void saludar(String nombre) {
    System.out.println("Hola, " + nombre + "!   Cmo   est s?");
}
```

En el método denominado “saludar” se recibe un parámetro de tipo String llamado “nombre”. Dentro del cuerpo del método, se imprime una cadena de saludo que incluye el valor del parámetro “nombre”. Al llamar a este método, debemos proporcionar un argumento de tipo String que se asignará al parámetro “nombre”.

2.4.3. Argumentos

Son los valores concretos que se pasan a un método cuando este es llamado. Estos valores corresponden a los parámetros definidos en el método. Una vez que hemos declarado un método, podemos llamarlo desde otro lugar de nuestro código. La llamada de un método se realiza utilizando su nombre seguido de paréntesis, y se pueden pasar los argumentos correspondientes si es necesario.[3]

```
int resultado = sumar(5, 3); // Llamada a un metodo con retorno
```

```
public int sumar(int a, int b) {
    return a + b;
}
```

2.5. HashMap

HashMap es una estructura de datos que implementa la interfaz Map y utiliza una tabla hash para almacenar pares clave-valor. Permite la inserción de elementos basándose en pares clave-valor y proporciona un rendimiento en tiempo constante para operaciones básicas como agregar o recuperar elementos. [4]

Los HashMaps son una estructura de datos versátil y ampliamente utilizada en Java, que ofrece acceso y manipulación eficiente de datos basados en claves únicas.

Algunos punto del HashMap pueden ser los siguiente:

- **Pares clase-valor**: HashMap almacena los elementos como pares clave-valor, donde cada clave es única. Las claves se utilizan para recuperar los valores correspondientes.
- **Recuperación rápida**: recuperar elementos de un HashMap por sus claves es eficiente porque calcula el código hash de la clave y obtiene directamente el valor correspondiente del índice calculado.
- **Claves y valores nulos**: HashMap permite una sola clave nula y varios valores nulos. Esto significa que una clave puede ser nula, pero no se permiten claves duplicadas, ya que deben ser únicas.
- **Mecanismo de hash**: Internamente, HashMap utiliza el hash para determinar la ubicación de almacenamiento de los elementos. Aplica una función hash a las claves para calcular sus códigos hash, convirtiéndolos en índices de matriz donde se almacenan los elementos.
- **Iteración**: es posible iterar a través de los elementos de un HashMap utilizando iteradores o bucles for mejorados.

Debido a sus efectivas capacidades de recuperación y a su técnica de almacenamiento de pares clave-valor, HashMap de Java es una estructura de datos flexible que puede usarse en una variedad de contextos. [4]

Permite un acceso rápido a los elementos según sus claves, independientemente de su posición dentro de la colección. Esta flexibilidad lo convierte en un elemento clave para diversas aplicaciones, desde la gestión de información de usuarios hasta la creación de tablas de búsqueda dinámicas. [4]

3. Desarrollo

3.1. Descripción de la Implementación

Se crearon 2 archivos, uno llamado Proyecto y otro llamado carrito. En el archivo llamado Proyecto lo que se focalizo fue la creación de un menú, esto se hizo con el uso de un **DO-WHILE** en el que se evalúa la condición y según lo que el usuario ingrese se realizara cierta acción. En el menú se presentan 4 opciones dentro de un switch que tendrán la función de llamar al método correspondiente

- Mostrar carrito
- Agregar producto
- Eliminar producto
- Salir

El archivo llamado Carrito representa un producto, el cual contiene dos atributos principales: el nombre y el precio. Estos se inicializan a través de un constructor, lo que permite instanciar diferentes productos con sus valores específicos.

Se redefine el método toString(), propio de Java, con el fin de que cada objeto pueda convertirse fácilmente a texto legible. En este caso, el producto se muestra con su nombre y su precio, en un formato adecuado para el usuario.

La clase también incluye métodos estáticos que permiten manipular un carrito de compras representado mediante un HashMap. Este tipo de estructura se eligió porque permite almacenar elementos en pares clave–valor, facilitando la búsqueda, inserción y eliminación de productos mediante el nombre como identificador único.

Los métodos implementados cumplen diferentes funciones:

- Mostrar los productos: Recorre los valores almacenados en el HashMap y los imprime de manera enumerada, apoyándose en el método sobrescrito de conversión a cadena.
- Agregar un producto: Inserta un nuevo producto en el HashMap utilizando su nombre como clave. Esto asegura que, si ya existe un producto con ese nombre, será reemplazado.
- Agregar un producto: Inserta un nuevo producto en el HashMap utilizando su nombre como clave. Esto asegura que, si ya existe un producto con ese nombre, será reemplazado.
- Eliminar un producto: Remueve del HashMap el producto cuyo nombre coincida con el solicitado, actualizando así el estado del carrito

3.2. Pruebas Realizadas

Para comenzar la prueba de el programa luego de compilar todo y ejecutar el archivo Proyecto, primero se coloco el número 2, que era el correspondiente a "*Agregar Producto*", al hacerlo nos pidió ingresar el nombre del producto que en esta prueba fue "*BANANA*", una vez ingresado nos pidió el precio del producto, en el cuál se coloco 10. Hicimos eso 3 veces con los siguientes productos y precios, "*COCOA*", 15, "*AZUCAR*", 35. Posterior a esto lo que hicimos fue probar la función de "*Mostrar Producto*"poniendo en la terminal en número 1 y nos arrojo la lista de los productos ingresados con su respectivo consto. Para finalizar hicimos uso del número 3 que es para eliminar un producto mediante su nombre, el producto que se eligió fue COCOA, al ingresar esa cadena en la terminal y luego mostrar los productos pudo notarse que se elimino correctamente. Para salir del programa solo colocamos en la terminal el número 4 y se finalizo el programa.

4. Resultados

```java
import java.util.HashMap;
import java.util.Scanner; //To be Implemented, Products are meant to be stored in a HashMap
public class Proyecto {
    Run main | Debug main
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
    HashMap<String, Carrito> Cart = new HashMap<>();

        int opcion = 0, cont = 0, price;
        String name;
        do{
            System.out.println("Bienvenido! Ingresa una de las siguientes opciones:");
            System.out.println("[1] - Mostrar Objetos\n[2] - Agregar Objetos\n[3] - Eliminar Objetos\n[4] - Salir");
            do{
                opcion = scan.nextInt();
                scan.nextLine();
            }while (opcion <0 || opcion>4);

            switch (opcion) {
                case 1:
                    if(cont == 0) System.out.println("Asegurate de haber ingresado al menos 1 objeto");
                    else Carrito.show(Cart);
                    break;
                case 2:
                    System.out.println("Ingresa el Nombre del Producto que Deseas Agregar: ");
                    name = scan.nextLine();
                    System.out.println("Ingresa el Precio del Producto que Deseas Agregar: ");
                    price = scan.nextInt();
                    Carrito item = new Carrito(name, price);
                    Carrito.Add(item, Cart);
                    cont++;
                    break;
                case 3:
                    if(cont == 0) System.out.println("Asegurate de haber ingresado al menos 1 objeto");
                    else Carrito.show(Cart);
                    System.out.println("Ingresa el Nombre del Producto que Deseas Eliminar:");
                    name = scan.nextLine();
                    Carrito.Delete(name, Cart);
                    break;
                case 4:
                    scan.close();
                    return;

                default:
                    break;
            }

        }while(true);
    }
}
```

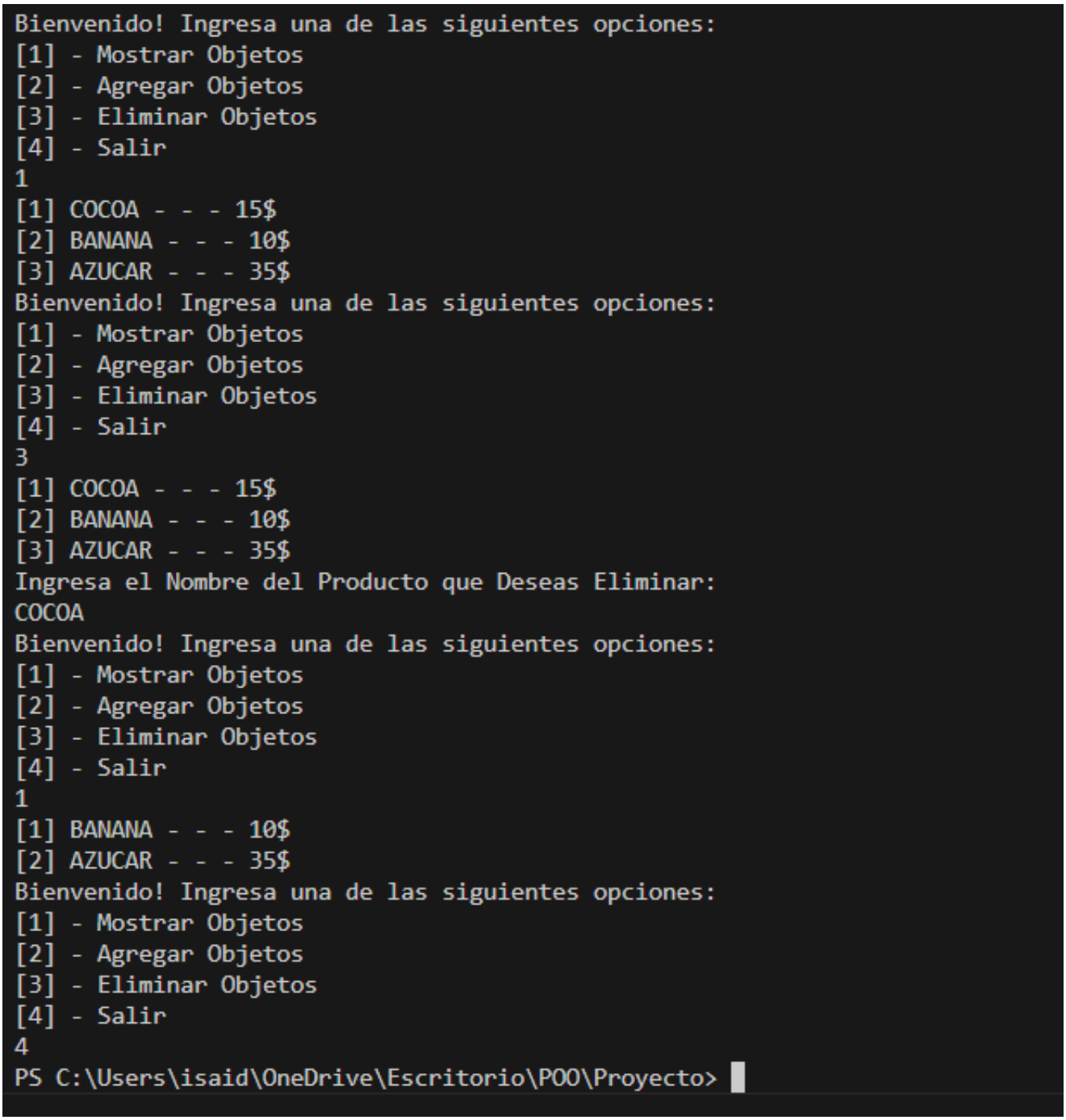
Aquí se muestra la clase principal en la cuál se lleva a cabo el menú, tratando los diferentes tipos de problemas que se pueden llegar a presentar a la hora de ingresar los datos.

```java
import java.util.HashMap;
public class Carrito{
    int precio;
    String name;
    public  Carrito(String name, int precio){
        this.name = name;
        this.precio = precio;
    }
    @Override
    public String toString() {
        return name + " - - - " + precio + "$";
    }

    public static void show(HashMap<String, Carrito> cart) {
        int i = 1;
        for (Carrito p : cart.values()) {
            System.out.println("[" + i + "] " + p);
            i++;
        }
    }

    public static boolean Add(Carrito toAdd, HashMap<String, Carrito> Cart){
    Cart.put(toAdd.name, toAdd);
           return true;
    }

    public static void Delete(String toDelete, HashMap<String, Carrito> Cart){
        Cart.remove(toDelete);
        return;
    }
}
```

En la imagen se pueden apreciar la clase carrito donde se declaran los métodos estáticos que se manejaran para la implementación de el carrito.

```
PS C:\Users\isaid\OneDrive\Escritorio\POO\Proyecto> java Proyecto
Bienvenido! Ingresa una de las siguientes opciones:
[1] - Mostrar Objetos
[2] - Agregar Objetos
[3] - Eliminar Objetos
[4] - Salir
2
Ingresa el Nombre del Producto que Deseas Agregar:
BANANA
Ingresa el Precio del Producto que Deseas Agregar:
10
Bienvenido! Ingresa una de las siguientes opciones:
[1] - Mostrar Objetos
[2] - Agregar Objetos
[3] - Eliminar Objetos
[4] - Salir
2
Ingresa el Nombre del Producto que Deseas Agregar:
COCOA
Ingresa el Precio del Producto que Deseas Agregar:
15
Bienvenido! Ingresa una de las siguientes opciones:
[1] - Mostrar Objetos
[2] - Agregar Objetos
[3] - Eliminar Objetos
[4] - Salir
2
Ingresa el Nombre del Producto que Deseas Agregar:
AZUCAR
Ingresa el Precio del Producto que Deseas Agregar:
35
Bienvenido! Ingresa una de las siguientes opciones:
[1] - Mostrar Objetos
[2] - Agregar Objetos
[3] - Eliminar Objetos
[4] - Salir
```

Se muestra como se fueron agregando los productos de prueba.



En esta imagen se muestran las demás funciones como mostrar carrito, eliminar producto y salir

5. Conclusiones

La practica permitió aplicar fundamentos de la Programación Orientada a Objetos mediante la creación de un programa simple que simula la función de un carrito de compras en la terminal. Se

logro diseñar y conectar dos clases Product y el Main (Test en este caso), estableciendo comunicación entre clases a través de objetos, definir atributos representativos como nombre y precio, y utilizar una estructura de datos HashMap para almacenar y gestionar los productos.

Durante el desarrollo del programa, se reforzaron conceptos importantes como la definición de constructores personalizados, la sobreescritura del método toString para presentar los datos de manera clara, y la implementación de métodos estáticos para realizar operaciones como agregar, eliminar y mostrar artículos. Estas técnicas demostraron cómo modularizar el código y mantenerlo organizado, lo que es fundamental para proyectos de mayor complejidad.

Además, la ejecución del programa en la terminal permitió observar de forma inmediata los resultados de las operaciones realizadas, lo que facilitó la verificación del correcto funcionamiento del sistema y el manejo adecuado de la estructura de datos. La práctica también destacó la importancia de la interacción con el usuario mediante menús y entradas de datos, reforzando la habilidad de diseñar programas intuitivos y funcionales.

Referencias

[1] DataCamp. (2025) Clases y objetos en java. Accedido: 16 de septiembre de 2025. [Online]. Available: https://www.datacamp.com/es/doc/java/classes-and-objects

[2] ——. (2025) Java constructors. Accedido: 16 de septiembre de 2025. [Online]. Available: https://www.datacamp.com/es/doc/java/constructors

[3] A. Barragán. (2023) Introducción a java: Métodos, parámetros y argumentos. Publicado: 24 de octubre de 2023; Accedido: 16 de septiembre de 2025. [Online]. Available: https://openwebinars.net/blog/introduccion-a-java-metodos-parametros-y-argumentos/

[4] A. Kumar. (2024) Guía de hashmap en java con ejemplos. Publicado: 18 de enero de 2024. [Online]. Available: https://builtin.com/articles/hashmap-in-java